Towards Update Relevance Checks in a Context Aware Mobile Information System

Hagen Höpfner
hoepfner@acm.org
International University in Germany
School of Information Technology
Campus 3, 76646 Bruchsal, Germany

Abstract: In order to reduce transmission cost mobile information system clients often cache data retrieved in a context aware manner. In the case of updates it may happen that this data becomes outdated. So, the caches must be invalidated. In this paper we discuss how to check the relevancy of updates regarding the server database as well as the client contexts.

1 Introduction and Motivation

Mobile information systems are often client/server systems with mobile clients that request information via a wireless connection. In order to reduce the communication costs, mobile devices cache received data. If data on the server changes, one has to inform the mobile clients that hold or should hold modified data. In [Hö05, HSS04] we presented approaches to check the relevancy of updates on the server side. We considered conjunctive queries with inequalities and focused on the relational data model. But mobile information systems have to consider the context of their usage. In [HS03a] we discussed a general context model that allows to hoard data automatically on the mobile client without formulating an explicit query. In this paper we present first ideas of combining both approaches having regard to the update relevancy check. The long-term objective is a context aware mobile information system, that handles conjunctive database queries with inequalities. The vision behind our work was presented in [HS03b].

The remainder of the paper is structured as follows. Section 2 is a summary of the foundations of this paper. In Section 3 we discuss the server side relevancy check for context aware conjunctive queries with inequalities. Finally the paper closes with a summary, conclusions and an outlook on ongoing research in Section 4.

2 Foundations

In this section we recapitulate some ideas of the used context model but restrict to aspects necessary for the understanding.

Parts of the databases on the server are enhanced by additional information (e.g. position, time). These extended parts are called fragments, and the additional information are called extensions v. The idea is, that the mobile client sends information about its context (e.g. "I am in London."), to the server. There they are used for selecting fragments, that relate to the mobile client. Therefore, we distinguish between server-extensions $v^{se}_{name} \in \mathcal{V}_{se}$ and client-extensions $v^{ce}_{name} \in \mathcal{V}_{ce}$. Both are vectors (e.g. two-dimensional geographic coordinates $v_{gc}(x, y); x, y \in \mathbb{R}$). The fragments on the server are extended by a set of server-extensions. The difference between client- and server-extensions is the usability of the parameters (in our last example x and y). On the base station parameter-functions are allowed for use as parameter. These functions compute the values of the parameters by querying the database. This way the redundant storage of data, that is already in the database, is avoided.

Fragments are almost arbitrary segments of one or more relation(s). Formally a fragment $\mathcal{F}$ is a tuple $\mathcal{F} = (f, V)$ with a fragmentation-function f and a set of server-extensions $V = \{v_0, \cdots, v_n\}$; $n \in \mathbb{N}$; $v_0, \cdots, v_n \in \mathcal{V}_{se}$. A fragmentation-function $f(r(R)) = r'(R')$ with $R' \subseteq R$ and $r'(R') \subseteq \pi_{R'}(r(R))$ is a query in the query-language used by the database management system on the host holding the database (e.g. SQL). In other words: A fragment is a view that was augmented by V.

Mobile clients use so called replication criteria for specifying their needs. A replication criterion is a triple and consists of a client-extension and of the minimal postulated and the maximal allowed tolerance of this criterion, e.g. $rk_{gc} = ((100, 200), 0, 20)$, means that the current position of the client is (100, 200) and the requested data must refer to a position in the circumference with the radius 20. The minimal postulated tolerance is useful if the mobile client already holds data and wants to extend its information horizon. In our example this could be done with $rk_{gc} = ((100, 200), 10, 20)$.

The decision, whether a fragment is of interest for the client or not, is done on the server using a ξ-function. If the result of this function is greater than or equal to the minimal postulated threshold and lower than or equal to the maximal allowed tolerance the respective fragment is taken into account. In our example such a ξ-function is the Euclidean distance $\xi_{gc}(v_{gc_1}, v_{gc_2}) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$. However, a ξ-function must not necessarily have the properties of a distance function, e.g., has not to be symmetric in the arguments.

Thus it appears that context aware database queries can be formulated as $CAQ = Q \wedge rk^1 \wedge rk^n$; $n \in \mathbb{N}$. In order to query the database, the identifying $name$ of each replication criterion must be included into the replication criterion values. For our geographical coordinates example[1], $CQA = \langle$`select * from concerts` $\wedge\, (gc, (100, 200), 0, 20)\rangle$ would result in all information of a fragment `concerts` related to the circumference 20 of the position (100,200). If the fragment `concerts` was enhanced by a time extension $v_{time} = (timevalue)$ too, the context aware query could look like $CAQ = \langle$`select * from concerts` $\wedge\, (gc, (100, 200), 0, 20) \wedge (time, (11\text{ am}), 0, 30)\rangle$. Assuming that the tolerance of the time criterion is given as minute values, the previous result would be additionally restricted to concert information regarding the time inter-

[1] For simplification we do not use our PSQ-notation of conjunctive queries here but refer to the possibility of converting PSQ to SQL [Hö05].

val $[10.30\text{ am}, 11.30\text{ am}]$. A database query Q may also include selections which must be a conjunction of predicates (in SQL `where` p_1 `and` $\cdots$ `and` p_2;). Assuming that our example fragment `concerts` contains the attribute genre a query could be $CAQ = \langle$`select * from concerts where genre='rock'` $\wedge\ (gc, (100, 200), 0, 20) \wedge (time, (11\text{ am}), 0, 30)\rangle$.

3 Update relecvancy checks

Once mobile clients retrieved the answer the data is cached locally. But what happens if the server data is modified? In our example a bringing forward of the beginning of a concert can happen. Now the caches on the clients that queried effected fragments are not longer valid and have to become invalidated. The first question is: *How can one detect clients effected by the update?*. On the other hand mobile clients are mobile. So, they may change their context. Assuming that clients submit their context periodically the second question is: *How to handle context updates?*. The third question that arises is: *What happens if the server-extensions changes?*. As we pointed out in Section 2, the server-extensions may use parameter-functions instead of values. So a database update could modify the fragment definitions as well. We discuss the three problems in the following.

3.1 Database Updates

First we assuming that the update does not effect the server-extensions but the database. In [Hö05, HSS04] we discussed a data centric approach to handle this problem. Queries form a Trie-index that indexes the IDs of the mobile clients. The idea is that queries that share a prefix of predicates are represented by the same sub-path in the Trie. The relevancy check traverses the Trie and computes check queries that are performed on the database. Sup-trees of nodes (predicates) that are recognized to be not effected by the update does not have to be taken into account in the next step. The result of this algorithm is a set of the IDs of the effected client. We will not discuss this algorithm here in more detail but refer to the original papers.

3.2 Context Updates

If the context of a mobile client has changed, answering the same database query might consider another fragment. So, one have to invalidate clients cache entries. The algorithm to check whether the context modification implies a fragment changeover or not is based on the replication criteria and the fragments extensions. The first step is to find the fragments used for answering the query under the old context. Therefore, we use a simple table, that

stores for each tuple[2] (CID, AID) the used replication criteria and the server-extensions of the fragments used for answering the query.

(CID, AID)	replication criteria	used server-extensions
$(1, 1)$	$(gc, (100, 200), 0, 20)$	$(gc,$ `SELECT x,y FROM geocoord WHERE city='London' and district='Chelsea'`$)$
$(2, 1)$	$(gc, (700, 500), 0, 10)$	$(gc,$ `SELECT x,y FROM geocoord WHERE city='London' and district='Greenwich'`$)$

Table 1: Query-Context-Index

Table 1 illustrates this Query-Context-Index $\mathcal{QCI}$. At this we have two clients that posted one query each and used various contexts. As mentioned in Subsection 3.1, the queries are stored in a Trie. So, context updates relevancy check can be handled by Algorithm 1.

Algorithmus 1: *Checking Context Update Relevancy*

ENSURE: *INPUT:* RK *// set of updated replication criteria*
CID *// ID of the client that changed its context*
$\mathcal{QCI}$ *// Query-Context-Index*

01. **for each** row **in** $\mathcal{QCI}$ **do**
02. **if** CID used in $row.column1$ **then** *// Client posted this query*
03. **for each** $rk = (v^{ce}_{name}, \Delta^{max}, \Delta^{min}) \in RK$ **do**
04. **if** $\exists v^{se}_{name} \in row.column3$ **then** *// compatible server extension found*
05. **if** $\Delta^{min} > \xi_{name}(v^{ce}_{name}, v^{se}_{name}) \vee \Delta^{max} < \xi_{name}(v^{ce}_{name}, v^{se}_{name})$ **then**
06. client holds outdated data, so notify client
07. **break**

3.3 Server-Extension Updates

The third update problem is the modification of data used in server-extensions. As mentioned in Section 2 and illustrated in Table 1 server-extensions may use parameter functions. Here it is an SQL-statement that fetches the required geographical coordinates from the database. So, we have to consider database updates that effect server-extensions. Obviously, checking the update relevancy for a parameter function is similar to Section 3.1

[2] CID is the ID of the mobile client and AID is the ID of the current query. At this, (CID, AID) must be unique.

but therefore, we need a "parameter function Trie". If an update is recognized to be relevant for an parameter function, we have to identify the queries that where answered from a fragment using this parameter function. Therefore, $\mathcal{QCI}$ can be used. The first step is to find such rows that contain the considered parameter function in column three. The next step is to check whether the replication criteria (stored in column two of the same row) selects the updated fragment or not. Therefore, the replication criteria must be reevaluated. Now, if the minimal postulated or the maximal allowed tolerance are violated, the client data is outdated and the client must be notified.

4 Summary, Conclusions and Outlook

In this paper we discussed first ideas to combine our approaches for checking the relevancy of updates for data cached on mobile clients with our context based query model. After a brief description of the foundations we discussed the kinds of updates that may occur in such a system and presented algorithms to handle them. The aim is, that mobile clients become notified if they hold outdated information.

The next step is to combine the both existing implementations in order to evaluate our combined approach. Furthermore we will consider incomplete contexts. The question here is, what happens if a fragment uses more or less or other server-extensions than replication criteria where given.

Remark: This paper does not include a section "related work" because we wanted to discuss our own ideas in more detail. However, please have a look at the referenced papers. We pigeonholed our research there.

References

[Hö05] Höpfner, H.: *Relevanz von Änderungen für Datenbestände mobiler Clients*. Dissertation. angenommen durch die Fakultät für Informatik der Otto-von-Guericke Universität Magdeburg. January 2005. in German.

[HS03a] Höpfner, H. and Sattler, K.-U.: Semantic Replication in Mobile Federated Information Systems. In: James, A., Conrad, S., and Hasselbring, W. (Eds.), *Proc. of EFIS2003, Coventry, UK 17th - 18th July, 2003*. pp. 36–41. Amsterdam. August 2003. Ios Press Inc.

[HS03b] Höpfner, H. and Sattler, K.-U.: SMoS: A Scalable Mobility Server. In: James, A. and Younas, M. (Eds.), *BNCOD20 Poster Proceedings, Coventry, UK 15th - 17th July, 2003*. pp. 49–52. School of Mathematical and Informational Sciences; Coventry University. July 2003.

[HSS04] Höpfner, H., Schosser, S., and Sattler, K.-U.: An Indexing Scheme for Update Notification in Large Mobile Information Systems. In: Lindner, W., Mesiti, M., Türker, C., Tzikzikas, Y., and Vakali, A. (Eds.), *Current Trends in Database Technology - EDBT 2004 Workshops: PhD, DataX, PIM, P2PDB, ClustWeb, Heraklion, Greece, March 14-18, 2004, Revised Papers*. volume 3268 of *LNCS*. pp. 345–354. Berlin. November 2004. Springer-Verlag.